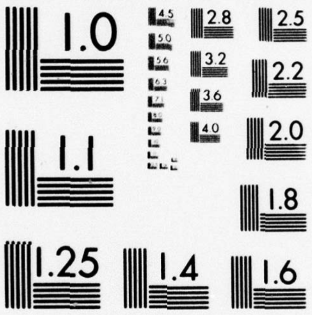
AD-A058 615 SRI INTERNATIONAL MENLO PARK CA F/G 5/7
A PROOF OF THE CORRECTNESS OF A SIMPLE PARSER OF EXPRESSIONS BY--ETC(U)
AUG 78 P Y GLOESS N00014-75-C-0816
UNCLASSIFIED SRI-TR-7 NL

| OF |
AD
A058615

END
DATE
FILMED
11-78
DDC



STANFORD RESEARCH INSTITUTE
Menlo Park, California 94025 · U.S.A.

(9) Technical Report N00014-75-C-0816-SRI-7

(11) Aug[redacted]78

12

(12) 54p.

# (6) A PROOF OF THE CORRECTNESS OF A SIMPLE PARSER OF EXPRESSIONS BY THE BOYER-MOORE SYSTEM.

(10) PAUL Y. GLOESS (14) SRI-TR-7

*Sponsored by:*

OFFICE OF NAVAL RESEARCH
DEPARTMENT OF THE NAVY
ARLINGTON, VIRGINIA 22217

ONR Contract Authority NR 049-378

(15) ONR Contract N00014-75-C-0816

SRI International
410281

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| | | |
|---|---|---|
| **REPORT DOCUMENTATION PAGE** | | *READ INSTRUCTIONS BEFORE COMPLETING FORM* |
| 1. REPORT NUMBER N00014-75-C-0816-SRI-7 | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle) A Proof of the Correctness of a Simple Parser of Expressions by the Boyer-Moore System | | 5. TYPE OF REPORT & PERIOD COVERED Technical Report |
| | | 6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s) Paul Y. Gloess | | 8. CONTRACT OR GRANT NUMBER(s) N00014-75-C-0816 ✓ |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS SRI International ✓ 333 Ravenswood Avenue Menlo Park, CA 94025 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR 049-378 |
| 11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research Department of the Navy Arlington, VA 22217 | | 12. REPORT DATE August 1978 / 13. NO. OF PAGES 51 |
| | | 15. SECURITY CLASS. (of this report) Unclassified |
| 14. MONITORING AGENCY NAME & ADDRESS (if diff. from Controlling Office) | | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this report)

Reproduction in whole or in part is permitted for any purpose of the United States Government. It may be released to the general public.

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

parser, program verification, theorem proving

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The objective of this report is to convey the essential idea of a proof by the Boyer-Moore Theorem Prover of the correctness of a parser. The proof required a total of 147 definitions and lemmas--all of which have been listed in the appendix.

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

78 09 08 039

19. KEY WORDS (Continued)

20 ABSTRACT (Continued)

Included in the following text are a description of the original problem submitted to the Theorem Prover and a sketch of the resultant proof, together with a discussion of the reasons that induced us to introduce some auxiliary functions. The report also contains the computer-generated proof of one of the main lemmas: INIT.SEG. The complete proof is available from the author.

## ABSTRACT

The objective of this report is to convey the essential idea of a proof by the Boyer-Moore Theorem Prover of the correctness of a parser. The proof required a total of 147 definitions and lemmas --all of which have been listed in the appendix.

Included in the following text are a description of the original problem submitted to the Theorem Prover and a sketch of the resultant proof, together with a discussion of the reasons that induced us to introduce some auxiliary functions. The report also contains the computer-generated proof of one of the main lemmas: INIT.SEG. The complete proof is available from the author.

## CONTENTS

## ACKNOWLEDGMENTS

We would like to thank Robert S. Boyer and J Strother Moore for their invaluable advice.

Our appreciation should also be expressed to the Theorem Prover for its very accurate proofs and incredible patience!

## I INTRODUCTION*

It seemed to us that a good way to learn something about the Boyer-Moore Theorem Prover was to submit an example to it. Although the specific nature of the example was of secondary importance to us in our selection, it was apparent from the outset that the example actually chosen presented potential difficulties. Our impression was indeed confirmed subsequently as we proceeded with the proof and found the example not merely fraught with difficulty, but intrinsically interesting.

Of principal interest is the fact that we wrote the example knowing little about the Theorem Prover --so little that none of our recursive functions could be proved as total. Nevertheless, we did succeed in obtaining one or two proofs based on induction.

As it was clear that the system was inherently averse to assuming totality and therefore exerted its efforts to make our proofs fail, we became especially concerned with the problem of totality. Upon reading Boyer and Moore's work on using the logic of recursive functions [2], we became convinced of the overriding importance of totality and understood what the system required for proving it. To fill the considerable gap between the original problem and the correctness hypothesis, we began introducing several auxiliary total functions and, consequently, a large number of lemmas. The hope that this procedure would lead to successful termination has indeed been borne out in practice.

There may conceivably exist a shorter path that could be utilized by the system toward its goal, i.e., proof of the correctness of our

--------

* The author was supported by a grant from IRIA, the French Institut de Recherche d'Informatique et d'Automatique, Domaine de Voluceau-Rocquencourt, 78150 LE CHESNAY.

**parser. If a shorter path is ultimately found, it will probably not require the inclusion of all these auxiliary functions. In any event, the particular path we have used is capable of successful termination --and we present it herein as conceived and performed by us.**

## II THE PROBLEM

The parser we are concerned with is a parser of expressions such as:

```
((A + B)*(- C)),
(A + (B + C)),
A,...
```

Note that our operators are either unary or binary, and that we require full bracketing and no more. For the parser, these are <u>nonexpressions</u>:

```
A + B,
(A + B + C),
- A,
((A + B)).
```

The <u>function</u> of the parser, called EVALEXPR, is to distinguish valid expressions from invalid, and to return a tree for every valid expression. For example a representation of the tree corresponding to the above first valid expression might be:

```
      *
    /   \
   +     -
  / \   /
 A   B C
```

To specify the correctness of the parser, we define a function PRINT (similar to the one used by INTERLISP to print lists). For any unary-binary tree such as the one above PRINT returns an expression. For the above tree the printed expression is the one already mentioned:

```
((A + B)*(- C)).
```

What we mean by correctness of the parser is this: "parsing the expression printed from every unary-binary tree restores the original tree". In a more formal way:

```
(TREEP X) => (EVALEXPR "NIL" (PRINT X)) = X.
```

This is the precise statement of the theorem we have proved for EVALEXPR*.

One can object that the above theorem does not signify the correctness of the parser, preferring the observation that "for every expression, the parser returns a tree that prints like the original expression". Alternatively, one might require the conjunction of both theorems.

However, it was our intention to prove only the first theorem. After doing so, we decided to name it "EVALEXPR.IS.CORRECT".

We shall now give our representation of trees and expressions and explain the definitions of TREEP, PRINT and EVALEXPR.**

## A. <u>LISP Representation of Trees and Expressions</u>

It is perhaps not yet clear to the reader that any significant difference exists between trees and expressions. If this were true there would be no real problem! In such examples as the proof of an optimizing compiler [3], trees and expressions are used synonymously, since one is only interested in their semantics. In the present context, however, we shall insist on a representation that differentiates between them.

We could have represented trees with a new shell by using the ADD.SHELL facility provided in the Theorem Prover. We thought that CONS was convenient enough, so we did not do this.

--------

* As we can see EVALEXPR takes 2 arguments: the first one plays only an internal role and when it is set to "NIL" the second one is the text to parse.

** All the functions or lemmas of our example are listed in the appendix.

The tree

```
          *
        /   \
      +       -
    / \     /
   A    B C
```

is represented by the S-expression

```
'((*) ((+) A B) ((-) C)),
```

and is, in fact, precisely

```
(CONS (CONS "*" "NIL")
      (CONS (CONS (CONS "+" "NIL")
                  (CONS "A" (CONS "B" "NIL")))
            (CONS (CONS (CONS "-" "NIL")
                        (CONS "C" "NIL"))
                  "NIL"))).
```

Note that we have adopted the convention that a tree should be a P-list (the CDR of a tree is "NIL" or a list)[*]. We also have adopted the convention of representing operators (such as "+", "*", ...) by lists of one element. Operators are thus distinguished from the terminals "A", "B", "C", ... represented by atoms. Although not absolutely necessary, this convention is consistent with our definition of expressions.

We also use lists to represent expressions. But here the length of the list is arbitrary. First of all, to avoid any confusion between the brackets of expressions and the lisp-brackets, we use "<" and ">" instead of "(" and ")". For example, the expression

```
((A + B) * (- C))
```

is represented by the S-expression

--------
[*] It is actually the Theorem Prover that made us alter our first definition of TREEP by showing us a counterexample (the parser returning only P-list trees).

```
   '(< < A (+) B > (*) < (-) C > > ),
```

and is precisely

```
(CONS "<"
    (CONS "<"
        (CONS "A"
            (CONS (CONS "+" "NIL")
                (CONS "B"
                    (CONS ">"
                        (CONS (CONS "*" "NIL")
                            (CONS "<"
                                (CONS (CONS "-" "NIL")
                                    (CONS "C"
                                        (CONS ">"
                                            (CONS ">"
                                                "NIL")))))))))))).
```

The Theorem Prover told us to exclude "<" from atoms. On the other hand "NIL" and even ">" are perfectly valid atoms. Thus:

```
   (("+") "NIL" ">")
```

is a valid tree and the corresponding expression

```
   ("<" "NIL" ("+") ">" ">")
```

is valid and well parsed by EVALEXPR, whereas

```
   ("<" "<" ("+") "A" ">")
```

is recognized as a nonexpression.*

Finally our predicate ATOMP for recognizing atoms is defined by

```
   (DEFN ATOMP(X)  (NLISTP X)&(X ≠ "<")).
```

Our predicate CONNECTP recognizes operators and is defined by

--------
* Although such special cases may seem relatively unimportant and are therefore easily overlooked, the Theorem Prover will remind the user of his negligence by noticing them immediately.

```
(DEFN CONNECTP(X) (LISTP X)).
```

Thus an operator might be any list, since we do not verify that it contains only one atom. We found that this test led to longer proofs, but that it was not essential to our purpose.

B. <u>The Function TREEP</u>

This predicate recognizes trees and should be obvious from the previous discussion:

```
(DEFN TREEP(X)
      (IF (NLISTP X)
          (ATOMP X)
          (IF (CONNECTP (TOPOFTREE X))
              (IF (UNARY X)
                  (TREEP (SUBTREE1 X))
                  (IF (BINARY X)
                      (AND (TREEP (SUBTREE1 X))
                           (TREEP (SUBTREE2 X)))
                      F))
              F))).
```

The nonrecursive functions TOPOFTREE, SUBTREE1, SUBTREE2, UNARY and BINARY are given in the appendix and their meanings should be clear. For example TOPOFTREE is CAR and SUBTREE2 is CADDR.

C. <u>The Function PRINT</u>

This function (see appendix) is very simple. If the input X is a nonlist, PRINT returns (CONS X "NIL"). If X is a unary tree, PRINT returns the concatenation

```
("<")||(TOPOFTREE X)||(PRINT (SUBTREE1 X))||(">"),
```

where (for readability) the "||" operator stands for APPEND, called CONCATCH throughout the appendix.

## D. The Parser EVALEXPR

EVALEXPR is the most complicated function in this problem. It takes two arguments L (Left) and R (Right). L is always set to "NIL" in an external call to EVALEXPR and R contains the text to parse.*

The algorithm we use is quite simple, but very inefficient. The main work of EVALEXPR is to try to decompose the input. Of course, when the text to parse is reduced to a list of one atom, nothing needs to be done. Otherwise the task of decomposition consists in finding the main operator of the expression, according to the pattern:

```
("<")||left_sub_exp_or_empty||main_op||right_sub_exp||(">").
```

When the element immediately following the initial "<" is an operator, decomposition has already been accomplished.

Otherwise EVALEXPR skips one element (moving to the right) by adding the first element of R to the end of L, as signified by the recursive call**

```
(EVALEXPR (APPEND L (CONS (CAR R) "NIL"))
          (CDR R)).
```

Then EVALEXPR will continue skipping until L and R are such that (CDR L) --i.e., left_sub_exp_or_empty-- is an expression and (CAR R) is an operator --i.e., main_op--. The search will eventually cease. For example, if (CDR L) happens to be an expression whereas (CAR R) is not an operator: EVALEXPR will return "<" to mean that L and R do not

--------

* The necessity of two arguments derives from the impossibility of defining a system of mutually recursive functions, as suggested by the classical LL(1) grammar of expressions.

** For greater clarity we have substituted APPEND, CAR, CONS and CDR in place of our CONCATCH, FIRSTCH, CHARCHAIN and REMAIN.

comprise an expression.*

Once the decomposition is obtained, the left part L starts with a "<" followed by either an empty list or an expression (called left_sub_exp_or_empty above); R starts with an operator (called main_op). So all EVALEXPR needs to do now is to:

* <u>check</u> that the last element of R is a closing bracket, i.e.:

  (LASTCH R) = ">";

* <u>check</u> that the elements between main_op and the last element of R form an expression, i.e.:

  [EVALEXPR "NIL" (BEGIN (CDR R))] ≠ "<";

* <u>return</u> either "<" or the adequate parse-tree obtained by CONSing the main operator, the parse-tree resulting from the left subexpression (if any), the one resulting from the right subexpression (which is always present), and "NIL" (to form a P-list).

EVALEXPR contains three different recursive calls. The first of these is

```
 (EVALEXPR (APPEND L (CONS (CAR R) "NIL"))
           (CDR R)),
```

which corresponds to skipping one element; the second is

```
 (EVALEXPR "NIL"
           (CDR L)),
```

for the left operand; the third call is

```
 (EVALEXPR "NIL"
           (BEGIN (CDR R))),
```

--------
* Until very late in our proofs, we had chosen "NIL" as a return value for nonexpressions; when we introduced the auxiliary predicate EXPRP and submitted a lemma relating EXPRP to EVALEXPR, the Theorem Prover found that ("NIL") was a valid expression according to EXPRP, but not with respect to EVALEXPR. So we chose "<" which is the only nonlist excluded by ATOMP.

corresponding to the right operand.

The totality of EVALEXPR stems from the fact that either the sum of the lengths of L and R decreases as in the second recursive call, or remains unchanged as in the first call; in the latter case the length of R decreases in both the first and third recursive calls.

Note that LASTCH (which returns the last element of a list) and BEGIN (which returns the list of all but the last elements of a list) are recursive functions. Therefore, some lemmas dealing with the manipulation of objects resembling character strings will be necessary.

## III SKETCH OF THE PROOF

The theorem we want to prove is:

```
EVALEXPR.IS.CORRECT:
(TREEP X) => (EVALEXPR "NIL" (PRINT X)) = X.
```

This conjecture suggests an induction on the variable X according to the schema of TREEP, which is what the system does. Three interesting cases are to be considered:*

* The base case: the tree X is reduced to a leaf,
* The unary case: X has only one branch, so that (PRINT X) is the list

  ```
  (APPEND (CONS "<" "NIL")
          (APPEND (CONS (TOPOFTREE X) "NIL")
                  (APPEND (PRINT (SUBTREE1 X))
                          (CONS ">" "NIL")))),
  ```

  constituting an expression of the form

  < operator operand >,

  which is easily parsed by EVALEXPR. The "pointer" moves only once to the right and the expression is immediately decomposed into its left part ("<") and right part operator||operand||(">"). The induction hypothesis

  (EVALEXPR "NIL" (PRINT (SUBTREE1 X))) = (SUBTREE1 X)

  and the expansion of (EVALEXPR "NIL" (PRINT X)) resolve this case.
* The binary case: here, of course, resides the entire difficulty of the problem. A simple expansion of EVALEXPR is insufficient, since the "pointer" must skip a variable

--------
* The Theorem Prover considers five cases, whereas the schema itself contains only two cases --the base case and the general case, which includes two induction hypotheses (corresponding to SUBTREE1 and SUBTREE2)--. The general case splits into four cases because the conjecture is an implication.

number (n+1) of symbols if n is the length of the subexpression that commences after the initial bracket. Precisely, (PRINT X) is here the list

```
(APPEND (CONS "<" "NIL")
        (APPEND (PRINT (SUBTREE1 X))
                (APPEND (CONS (TOPOFTREE X) "NIL")
                        (APPEND (PRINT (SUBTREE2 X))
                                (CONS ">" "NIL"))))).
```

Therefore, the proof consists in showing that the "pointer" must skip until past (PRINT (SUBTREE1 X)) (which is an expression by virtue of the induction hypothesis and the fact that X, being a tree, cannot be "<"). In other words, (PRINT (SUBTREE1 X)) must be the smallest subexpression starting after the initial "<".

Finally the proof of the whole problem is mainly based on the following lemma:

"Any proper initial segment of an expression is not itself an expression."

Let us call this the "initial-segment lemma," the computer proof of which is presented in Section V.

This kind of generalization requires the creation of auxiliary functions. Consequently, it is far beyond the system's capacity.

## IV THE HANDWRITTEN AUXILIARY FUNCTIONS

After several unsuccessful attempts it was evident that all the lemmas in the world could not solve our problem.

Without a significant generalization (involving new functions), the intrinsic schema of EVALEXPR could never be used. The general induction principle of Boyer and Moore [1] requires that all measured arguments of a function call whose schema contributes to the induction be variables.

Furthermore, the function EVALEXPR appeared especially difficult for two reasons. First, as we saw in subsection D of Section II, it contains three different recursive calls, but one of these --namely the shift of one position to the right-- seems to precede the other two chronologically. The second negative aspect is that some of the recursive calls "govern" (see [1]) others.

All these reasons led us to introduce the function CUT, then INCLUP and, finally, EXPRP. Although this idea did not emerge easily, at least for the first function, it appears now that these functions have a common characteristic: each of them reflects a specific part of the complex schema of EVALEXPR and is thus "simpler" than EVALEXPR.

### A. <u>Decomposition of an Expression by the Function CUT</u>

The goal achieved by this function is to move the "pointer" to the right until it splits the input expression into two segments

```
< left right
      ^
```

whose CONS is returned as the value of CUT. The "pointer" stops (in the general case where the expression begins with a "<") as soon as "left" is the empty list and "right" begins with an operator (unary case), or "left" is an expression (binary case). Note that CUT does not check the right part, except to stop the "pointer" in the unary case.

The only recursive call contained in CUT's definition is

```
   (CUT (APPEND L (CAR R))
        (CDR R)),
```

where L and R are the arguments.

The most interesting property of CUT is

```
CUT.HELPS.EVALEXPR:
   (EQUAL (EVALEXPR (CAR (CUT L R))
                    (CDR (CUT L R)))
          (EVALEXPR L
                    R)).
```

The proof of EVALEXPR.IS.CORRECT is thus reduced to the proof of

```
(EQUAL (CUT "NIL"
            (APPEND (CONS "<" "NIL")
                    (APPEND EXP1
                            (APPEND (CONS OP "NIL")
                                    (APPEND EXP2
                                            (CONS ">" "NIL"))))))
       (CONS (CONS "<" EXP1)
             (CONS OP (APPEND EXP2 (CONS ">" "NIL"))))),
```

where EXP1 and EXP2 are expressions and OP is an operator.

## B. The Ordering Relation INCLUP

This function takes the four arguments L1, R1, L2 and R2 and checks a logical relation between the ordered pair <L1,R1> and the ordered pair <L2,R2>, as best expressed by the following schema:

```
     L1          R1
   |----|-----------------|

   |-------|--------------|
     L2          R2
```

L1, R1, L2, R2 are lists. The relation (INCLUP L1 R1 L2 R2) holds when

(APPEND L1 R1) = (APPEND L2 R2)

and L1 is an initial segment of L2. However, INCLUP is not defined that way; it uses the same recursive call as CUT.

Of course, the INCLUP relation holds between CUT's arguments and CUT's result, as signified by the lemma

```
INCLUP.LR.CUTLR:
  (INCLUP L R
          (CAR (CUT L R)) (CDR (CUT L R))).
```

This lemma, joined to the preceding ones, reduces the proof of EVALEXPR.IS.CORRECT to that of the central lemma we formulated in Section III: "Any proper initial segment of an expression is not itself an expression."

## C. <u>The Initial-Segment Lemma and the Function EXPRP</u>

Let us remind the reader that EVALEXPR assigns to nonexpressions only the value "<". Hence, a possible statement of the initial-segment lemma is:

```
(IMPLIES (AND (NOT (EQUAL (EVALEXPR L (APPEND R S))
                          "<"))
              (LISTP S))
         (EQUAL (EVALEXPR L R)
                "<")).
```

This is actually a slightly more generalized form of the lemma, since "NIL" has been replaced by the variable L. Although it allows an induction using the schema of EVALEXPR, the proof fails. The essential reason for this is that EVALEXPR does not suggest the right induction --at least the one we had in mind. Furthermore, it seems that the argument L, however necessary from the induction viewpoint, is misleading. So we decided to define another function: EXPRP. As we shall see, EXPRP negates the first argument, L.

The proof we conceived of and subsequently checked with the system is as follows. Suppose that S is a list, and R and R¦¦S are both expressions. It is obvious that R has to start with a "<". It can be proved that each decomposition --of R and R¦¦S-- will have the same left part i.e., the lemma CUT.CONCATCH. Since R is an expression, the search for the main operator will stop at least two elements before the end (see lemma TWO.ELEMENTS), because the right part must start with an operator and finish with a ">". Hence, since we know that this search does not depend on the right part (except, perhaps, for the first element which is the main operator [see subsection A]), the search will certainly not depend on S, which is after the final ">" of R. So we have the situation:

```
 R: < left_part operator right_sub_exp >
R¦¦S: < left_part operator right_sub_exp_and_more >.
```

right_sub_exp_and_more must be an expression, but this is not possible (by induction hypothesis), since right_sub_exp is an expression and a proper initial segment of right_sub_exp_and_more.

Note that the variable L plays no role in this proof. At first glance the induction

```
  (p right_sub_exp S) => (p R S),
```

on R only, seems sufficient.

A second more careful look shows that the variable S of the induction hypothesis needs to be instantiated, according to the schema:

```
  (p right_sub_exp (CONS ">" (BEGIN S))) => (p R S),
```

so as to take into account the final bracket of R, but omit that of R¦¦S[*].

As the system does not instantiate those variables that do not participate in the induction, we used an artifice to force the

--------
[*] Remember that BEGIN returns the list S minus its last element.

instantiation of S in the induction hypothesis: the adjunction of the dummy argument DUMMY to the auxiliary function EXPRP. Otherwise R would have been its only argument.

The only recursive call contained in the definition of EXPRP is

```
  (EXPRP (BEGIN (CDR (CDR (CUT "NIL" R))))
         (CONS ">" (BEGIN DUMMY))),
```

where the first argument represents the right subexpression of R when R is an expression. Note that EXPRP does not refer to the left subexpression of R, whereas EVALEXPR does. Also note that EXPRP does not depend on DUMMY (see appendix).

Expressed in terms of EXPRP, the initial-segment lemma becomes

```
(IMPLIES (AND (EXPRP (APPEND R S) S)
              (LISTP S))
         (NOT (EXPRP R S))).
```

Naturally we had to prove that EXPRP recognizes the same set of expressions as EVALEXPR, i.e., the lemma:

```
EXPRP.EQUAL.EVALEXPR:
  (EQUAL (EXPRP R DUMMY)
         (NOT (EQUAL (EVALEXPR "NIL" R)
                     "<"))).
```

## V COMPUTER-PROOF OF THE INITIAL-SEGMENT LEMMA

The system has proved a contrapositive form of the initial-segment lemma:

```
INIT.SEG:
  [IMPLIES (AND (LISTP B)
                (EXPRP A B))
           (NOT (EXPRP (CONCATCH A B) B].
```

Remember that :

* CONCATCH is a synonym of APPEND
* LASTCH returns the last element of a list
* BEGIN returns the input list minus its last element.

The beginning of the proof shows what induction schema was chosen from the function EXPRP. The schema contains five base cases and one general case. In the remainder of the proof the five base cases split into six, numbered from eight to three. The general case bisects into cases no. 2 and no. 1, because the induction hypothesis is an implication. Case 2 is comparatively barren. Case 1 is the interesting one of the pair.

From this point, the system first expands the hypothesis "(EXPRP A B)", which is not useful and necessitates the additional lemma REBUILD.EXPRP.FROM.EXPANSION.

Then the hypothesis resulting from the induction (i.e., the conclusion of the induction hypothesis) is transformed into

```
  (NOT (EXPRP (CONCATCH (CDR (CDR (CUT "NIL" A)))
                        (BEGIN B))
              (CONS ">" (BEGIN B)))),
```

owing to the four linked technical lemmas CONCATCH.BEGIN.CONS.LASTCH, TWO.ELEMENTS, LASTCH.CUT and LISTP.CDR.CUT given in the appendix. For example, LASTCH.CUT asserts that the last element of "(CUT L R)" is the last element of R if R is a list.

The next step is the elimination, as usual, of (CAR A) and (CDR A).

Finally, the function EXPRP in the conclusion expands by virtue of the lemma EXPRP.CONCATCH.EXPANDS. The other lemmas cited contribute to the new form of the conclusion,

```
  (OR conclusion_of_induction_hypothesis
      (NOT (EQUAL (LASTCH B) ">"))),
```

which is not visible since this conjecture is immediately reduced to (TRUE).

Note that the lemma CUT.CONCATCH (already mentioned in subsection C of Section IV), although not cited in the present proof, plays an important role in the transformation of the conclusion. This lemma affirms the equality of the left subexpressions of expressions A and (CONCATCH A B), i.e., in a slightly stronger form:

```
CUT.CONCATCH :
  (IMPLIES (NOT (EQUAL (EVALEXPR U A) "<"))
           (EQUAL (CUT U (CONCATCH A B))
                  (CONS (CAR (CUT U A))
                        (CONCATCH (CDR (CUT U A))
                                  B)))).
```

The computer proof of INIT.SEG starts on the next page. All cited lemmas (except for the basic axioms dealing with the shell [CONS, CAR, CDR, LISTP]) and functions are listed in the appendix.

```
_PROVE.LEMMA(INIT.SEG (REWRITE)
          (IMPLIES (AND (LISTP B) (EXPRP A B))
                   (NOT (EXPRP (CONCATCH A B) B)))
          NIL NIL)
Name this *1.
```

Of course, when all else fails (and it has), induct on something!

There are two plausible inductions. They don't merge. And, neither is clean! And they appear equally likely. So the one which will probably lead to eliminating the nastiest expression was picked. We will induct according to the following schema:

```
     (AND (IMPLIES (NOT (LISTP (CDR (CUT "NIL" A))))
                   (p A B))
          (IMPLIES (NOT (LISTP A)) (p A B))
          (IMPLIES (NOT (LISTP (CDR A)))
                   (p A B))
          (IMPLIES (NOT (EQUAL (CAR A) "<"))
                   (p A B))
          (IMPLIES (NOT (LISTP (CAR (CDR (CUT "NIL" A)))))
                   (p A B))
          (IMPLIES (AND (LISTP (CDR (CUT "NIL" A)))
                        (LISTP A)
                        (LISTP (CDR A))
                        (EQUAL (CAR A) "<")
                        (LISTP (CAR (CDR (CUT "NIL" A))))
                        (p (BEGIN (CDR (CDR (CUT "NIL" A))))
                           (CONS ">" (BEGIN B))))
                   (p A B))).
```

The inequalities BEGIN.CDR.LNG.LESSP, CDR.CUT.LESSEQP and EXPRP.IS.TOTAL establish that the measure (LNG A) decreases according to the well-founded function LESSP in the induction step of the schema. This produces eight new conjectures:

```
 8|(IMPLIES (AND (NOT (LISTP (CDR (CUT "NIL" A))))
  |              (LISTP B)
  |              (EXPRP A B))
  |          (NOT (EXPRP (CONCATCH A B) B))),
  |which simplifies, using the lemmas CAR.NLISTP,
  |CAR.CONCATCH.LISTP and CDR.CONCATCH.LISTP, and unfolding
  |EXPRP and CONCATCH, to:

   *|(TRUE).

 7|(IMPLIES (AND (NOT (LISTP A))
  |              (LISTP B)
  |              (EXPRP A B))
  |          (NOT (EXPRP (CONCATCH A B) B))).
  |This simplifies, expanding the definition of EXPRP, to:
```

```
 *¦(TRUE).

6¦(IMPLIES (AND (NOT (LISTP (CDR A)))
 ¦               (LISTP B)
 ¦               (EXPRP A B))
 ¦          (NOT (EXPRP (CONCATCH A B) B))),
 ¦which simplifies, applying the lemmas CAR.CONS and
 ¦CDR.CONS, and expanding the definitions of EXPRP and
 ¦CONCATCH, to:

 *¦(TRUE).

5¦(IMPLIES (AND (NOT (EQUAL (CAR A) "<"))
 ¦               (LISTP B)
 ¦               (EXPRP A B))
 ¦          (NOT (EXPRP (CONCATCH A B) B))),
 ¦which simplifies, rewriting with CAR.CONCATCH.LISTP and
 ¦CDR.CONCATCH.LISTP, and expanding the definitions of
 ¦EXPRP and CONCATCH, to:

 *¦(TRUE).

4¦(IMPLIES (AND (NOT (LISTP (CAR (CDR (CUT "NIL" A)))))
 ¦               (LISTP B)
 ¦               (EXPRP A B))
 ¦          (NOT (EXPRP (CONCATCH A B) B))),
 ¦which we can simplify, applying CAR.CONCATCH.LISTP and
 ¦CDR.CONCATCH.LISTP, and expanding the functions EXPRP
 ¦and CONCATCH, to:

 *¦(TRUE).

3¦(IMPLIES (AND (LISTP (CDR (CUT "NIL" A)))
 ¦               (LISTP A)
 ¦               (LISTP (CDR A))
 ¦               (EQUAL (CAR A) "<")
 ¦               (LISTP (CAR (CDR (CUT "NIL" A))))
 ¦               (NOT (LISTP (CONS ">" (BEGIN B))))
 ¦               (LISTP B)
 ¦               (EXPRP A B))
 ¦          (NOT (EXPRP (CONCATCH A B) B))),
 ¦which we can simplify, using the lemma LISTP.CDR.CUT, to:

 *¦(TRUE).

2¦(IMPLIES
 ¦      (AND (LISTP (CDR (CUT "NIL" A)))
 ¦           (LISTP A)
 ¦           (LISTP (CDR A))
 ¦           (EQUAL (CAR A) "<")
 ¦           (LISTP (CAR (CDR (CUT "NIL" A))))
 ¦           (NOT (EXPRP (BEGIN (CDR (CDR (CUT "NIL" A))))
|                     (CONS ">" (BEGIN B))))
|           (LISTP B)
|           (EXPRP A B))
|      (NOT (EXPRP (CONCATCH A B) B))).
|This simplifies, rewriting with LISTP.CDR.CUT, and
|unfolding EXPRP, to:

 *|(TRUE).

1|(IMPLIES
 | (AND
 |  (LISTP (CDR (CUT "NIL" A)))
 |  (LISTP A)
 |  (LISTP (CDR A))
 |  (EQUAL (CAR A) "<")
 |  (LISTP (CAR (CDR (CUT "NIL" A))))
 |  (NOT
 |      (EXPRP (CONCATCH (BEGIN (CDR (CDR (CUT "NIL" A))))
 |                       (CONS ">" (BEGIN B)))
 |             (CONS ">" (BEGIN B))))
 |  (LISTP B)
 |  (EXPRP A B))
 | (NOT (EXPRP (CONCATCH A B) B))),
|which we can simplify, rewriting with LISTP.CDR.CUT and
|LASTCH.CDR.LISTP.ELIM, and expanding the definitions of
|LASTCH, EXPRP and CONCATCH, to:

  1|(IMPLIES
   | (AND
   |  (LISTP A)
   |  (LISTP (CDR A))
   |  (EQUAL (CAR A) "<")
   |  (LISTP (CAR (CDR (CUT "NIL" A))))
   |  (NOT
   |    (EXPRP (CONCATCH (BEGIN (CDR (CDR (CUT "NIL" A))))
   |                     (CONS ">" (BEGIN B)))
   |           (CONS ">" (BEGIN B))))
   |  (LISTP B)
   |  (EXPRP (BEGIN (CDR (CDR (CUT "NIL" A))))
   |         (CONS ">" (BEGIN B)))
   |  (EQUAL (LASTCH (CDR A)) ">"))
   | (NOT (EXPRP (CONS (CAR A) (CONCATCH (CDR A) B))
   |             B))),
   |which we can again simplify, applying
   |CONCATCH.BEGIN.CONS.LASTCH, TWO.ELEMENTS, LASTCH.CUT,
   |LISTP.CDR.CUT and LASTCH.CDR.LISTP.ELIM, and expanding
   |LASTCH, to:

    1|(IMPLIES
     | (AND
     |     (LISTP A)
     |     (LISTP (CDR A))
```

```
¦     (EQUAL (CAR A) "<")
¦     (LISTP (CAR (CDR (CUT "NIL" A))))
¦     (NOT (EXPRP (CONCATCH (CDR (CDR (CUT "NIL" A)))
¦                           (BEGIN B))
¦                 (CONS ">" (BEGIN B))))
¦     (LISTP B)
¦     (EXPRP (BEGIN (CDR (CDR (CUT "NIL" A))))
¦            (CONS ">" (BEGIN B)))
¦     (EQUAL (LASTCH (CDR A)) ">"))
¦ (NOT (EXPRP (CONS (CAR A) (CONCATCH (CDR A) B))
¦             B))).
¦Eliminate the undesirable expressions (CDR A) and
¦(CAR A), by using the lemma CAR/CDR.ELIM backwards
¦and then generalizing.  We thus obtain:

 1¦(IMPLIES
  ¦ (AND
  ¦  (LISTP (CONS Z X))
  ¦  (LISTP X)
  ¦  (EQUAL Z "<")
  ¦  (LISTP (CAR (CDR (CUT "NIL" (CONS Z X)))))
  ¦  (NOT
  ¦   (EXPRP
  ¦     (CONCATCH (CDR (CDR (CUT "NIL" (CONS Z X))))
  ¦               (BEGIN B))
  ¦     (CONS ">" (BEGIN B))))
  ¦  (LISTP B)
  ¦  (EXPRP
  ¦       (BEGIN (CDR (CDR (CUT "NIL" (CONS Z X)))))
  ¦       (CONS ">" (BEGIN B)))
  ¦  (EQUAL (LASTCH X) ">"))
  ¦ (NOT (EXPRP (CONS Z (CONCATCH X B)) B))).
  ¦This can be further simplified, applying the
  ¦lemmas CDR.NLISTP, CAR.CONS, CDR.CONS,
  ¦EXPRP.CONCATCH.EXPANDS,
  ¦REBUILD.EXPRP.FROM.EXPANSION,
  ¦LISTP.CAR.CDRCUT.CONCATCH and
  ¦EXPRP.CONCATCH.EQUAL.COMMON, and expanding the
  ¦functions CONCATCH, EVALEXPR and CUT, to:

   *¦(TRUE).

Ha!  That finishes the proof of *1.


Q.E.D.


Load average during proof:  2.855174
Elapsed time:  271.84 seconds
CPU time (devoted to theorem proving):  99.956 seconds
```

```
IO time:  2.802 seconds
CONSes consumed:  122298


PROVED
```

# Appendix A

# FUNCTIONS AND LEMMAS

## Appendix A

## FUNCTIONS AND LEMMAS

```
(FILECREATED " 1-Jun-78 03:13:12" <GLOESS>HISTO..51 28548

     changes to:  THM

     previous date: "30-May-78 22:41:54" <GLOESS>HISTO..50)


(PRETTYCOMPRINT HISTOCOMS)

(RPAQQ HISTOCOMS (HISTORY.STATUS THM))

(RPAQQ HISTORY.STATUS NIL)

(RPAQQ THM ((PROVE.LEMMA SUB1X.LESSP.PLUSYX (REWRITE)
                         (IMPLIES (AND (NUMBERP X)
                                       (NUMBERP Y)
                                       (NOT (EQUAL X 0)))
                                  (LESSP (SUB1 X)
                                         (PLUS Y X)))
                         NIL NIL)
        (PROVE.LEMMA CAR.LESSEQP (INDUCTION)
                     (IMPLIES (NOT (EQUAL (COUNT (CAR X))
                                          (COUNT X)))
                              (LESSP (COUNT (CAR X))
                                     (COUNT X)))
                     NIL NIL)
        (PROVE.LEMMA CDR.LESSEQP (INDUCTION)
                     (IMPLIES (NOT (EQUAL (COUNT (CDR X))
                                          (COUNT X)))
                              (LESSP (COUNT (CDR X))
                                     (COUNT X)))
                     NIL NIL)
        (DEFN NLISTP (X)
              (NOT (LISTP X))
              NIL)
        (DEFN NULLP (X)
              (EQUAL X "NIL")
              NIL)
        (DEFN LEFTPAR NIL "<" NIL)
        (DEFN LEFTPARP (X)
              (EQUAL X (LEFTPAR))
              NIL)
        (DEFN ATOMP (X)
              (IF (NLISTP X)
            (IF (LEFTPARP X)
                F T)
            F)
        NIL)
(DEFN CONNECTP (X)
        (LISTP X)
        NIL)
(DEFN TOPOFTREE (X)
        (CAR X)
        NIL)
(DEFN UNARY (X)
        (IF (NLISTP (CDR X))
            F
            (NULLP (CDDR X)))
        NIL)
(DEFN SUBTREE1 (X)
        (CADR X)
        NIL)
(DEFN BINARY (X)
        (UNARY (CDR X))
        NIL)
(DEFN SUBTREE2 (X)
        (CADDR X)
        NIL)
(DEFN
    TREEP
    (X)
    (IF (NLISTP X)
        (IF (ATOMP X)
            T F)
        (IF (CONNECTP (TOPOFTREE X))
            (IF (UNARY X)
                (TREEP (SUBTREE1 X))
                (IF (BINARY X)
                    (AND (TREEP (SUBTREE1 X))
                         (TREEP (SUBTREE2 X)))
                    F))
            F))
    NIL)
(DEFN CHARCHAIN (X)
        (CONS X "NIL")
        NIL)
(DEFN CONCATCH (X Y)
        (IF (NLISTP X)
            Y
            (CONS (CAR X)
                  (CONCATCH (CDR X)
                            Y)))
        NIL)
(DEFN RIGHTPAR NIL ">" NIL)
(DEFN
    PRINT
```

```
  (X)
  (IF
    (NLISTP X)
    (CHARCHAIN X)
    (IF (UNARY X)
        (CONCATCH (CHARCHAIN (LEFTPAR))
                  (CONCATCH (CHARCHAIN (TOPOFTREE X))
                            (CONCATCH (PRINT (SUBTREE1 X))
                                      (CHARCHAIN (RIGHTPAR)))))
        (CONCATCH (CHARCHAIN (LEFTPAR))
                  (CONCATCH
                    (PRINT (SUBTREE1 X))
                    (CONCATCH (CHARCHAIN (TOPOFTREE X))
                              (CONCATCH (PRINT (SUBTREE2 X))
                                        (CHARCHAIN (RIGHTPAR))))
                    ))))
  NIL)
(DEFN FIRSTCH (X)
      (CAR X)
      NIL)
(DEFN REMAIN (X)
      (CDR X)
      NIL)
(DEFN G (X)
      X NIL)
(DEFN ONEORTWO (X Y)
      (IF (LEFTPARP X)
          (CONS Y "NIL")
          (CONS X (CONS Y "NIL")))
      NIL)
(DEFN LASTCH (X)
      (IF (NLISTP (REMAIN X))
          (FIRSTCH X)
          (LASTCH (REMAIN X)))
      NIL)
(PROVE.LEMMA LASTCH.CONS (REWRITE)
             (IMPLIES (LISTP Y)
                      (EQUAL (LASTCH (CONS X Y))
                             (LASTCH Y)))
             NIL NIL)
(DEFN PLISTP (X)
      (IF (LISTP X)
          (PLISTP (CDR X))
          (EQUAL X "NIL"))
      NIL)
(PROVE.LEMMA LASTCH.CONCATCH (REWRITE)
             (IMPLIES (AND (LISTP U)
                           (LISTP V))
                      (EQUAL (LASTCH (CONCATCH U V))
                             (LASTCH V)))
             NIL NIL)
(PROVE.LEMMA LISTP.CONCATCH (REWRITE)
             (IMPLIES (LISTP X)
                      (LISTP (CONCATCH X Y)))
             NIL NIL)
(PROVE.LEMMA CONCATCH.PLISTP (REWRITE GENERALIZE)
             (IMPLIES (PLISTP B)
                      (PLISTP (CONCATCH A B)))
             NIL NIL)
(DEFN INCLUP (L1 R1 L2 R2)
      (IF (EQUAL L1 L2)
          (EQUAL R1 R2)
          (IF (NLISTP R1)
              F
              (INCLUP (CONCATCH L1 (CONS (CAR R1)
                                         "NIL"))
                      (CDR R1)
                      L2 R2)))
      NIL)
(PROVE.LEMMA INCLUP.LISTPL1.NLISTPL2 (REWRITE)
             (IMPLIES (AND (LISTP L1)
                           (NOT (LISTP L2)))
                      (NOT (INCLUP L1 R1 L2 R2)))
             NIL NIL)
(PROVE.LEMMA INCLUP.CDRL1.CDRL2 (REWRITE)
             (IMPLIES (AND (LISTP L1)
                           (LISTP L2)
                           (EQUAL (CAR L1)
                                  (CAR L2)))
                      (EQUAL (INCLUP L1 R1 L2 R2)
                             (INCLUP (CDR L1)
                                     R1
                                     (CDR L2)
                                     R2)))
             NIL NIL)
(PROVE.LEMMA INCLUP.CONCATCH (REWRITE)
             (IMPLIES (AND (PLISTP D)
                           (PLISTP L))
                      (INCLUP L (CONCATCH D R)
                                (CONCATCH L D)
                                R))
             NIL NIL)
(PROVE.LEMMA INCLUP.CONCATCH.CROK (REWRITE)
             (IMPLIES (AND (EQUAL (CONCATCH Z "NIL")
                                  Z)
                           (EQUAL (INCLUP (CONCATCH Z A)
                                          R1
                                          (CONCATCH Z "NIL")
                                          R2)
                                  (INCLUP A R1 "NIL" R2)))
                      (EQUAL (INCLUP (CONCATCH Z A)
                                     R1 Z R2)
                             (INCLUP A R1 "NIL" R2)))
             NIL NIL)
```

```
(PROVE.LEMMA INCLUP.CONCATCH.ELIM (REWRITE)
             (EQUAL (INCLUP (CONCATCH Z A)
                            R1
                            (CONCATCH Z B)
                            R2)
                    (INCLUP A R1 B R2))
             NIL NIL)
(PROVE.LEMMA CONCATCH.NIL.LOGICAL (REWRITE)
             (EQUAL (EQUAL (CONCATCH Z "NIL")
                           Z)
                    (PLISTP Z))
             NIL NIL)
(PROVE.LEMMA INCLUP.CONCATCH.ELIM.NIL (REWRITE)
             (IMPLIES (PLISTP Z)
                      (EQUAL (INCLUP (CONCATCH Z A)
                                     R1 Z R2)
                             (INCLUP A R1 "NIL" R2)))
             NIL NIL)
(PROVE.LEMMA INCLUP.IS.TRANSITIVE (REWRITE)
             (IMPLIES (AND (PLISTP L3)
                           (INCLUP L1 R1 L2 R2)
                           (INCLUP L2 R2 L3 R3))
                      (INCLUP L1 R1 L3 R3))
             NIL NIL)
(DEFN BEGIN (X)
      (IF (NLISTP (REMAIN X))
          "NIL"
          (CONS (FIRSTCH X)
                (BEGIN (REMAIN X))))
      NIL)
(PROVE.LEMMA BEGIN.LESSP (INDUCTION)
             (IMPLIES (LISTP X)
                      (LESSP (COUNT (BEGIN X))
                             (COUNT X)))
             NIL NIL)
(PROVE.LEMMA BEGIN.CONCATCH (REWRITE)
             (IMPLIES (PLISTP U)
                      (EQUAL (BEGIN (CONCATCH U (CONS V "NIL")))
                             U))
             NIL NIL)
(PROVE.LEMMA TREEP.IMPLIES.PLISTP.PRINT (GENERALIZE REWRITE)
             (IMPLIES (TREEP X)
                      (PLISTP (PRINT X)))
             NIL NIL)
(DEFN RIGHTPARP (X)
      (EQUAL X (RIGHTPAR))
      NIL)
(DEFN LNG (X)
      (IF (NLISTP X)
          0
          (ADD1 (LNG (CDR X))))
      NIL)
```

```
(DEFN LNG2 (X Y)
      (PLUS (LNG X)
            (LNG Y))
      NIL)
(PROVE.LEMMA LNG2.EXPR.GOES.DOWN (INDUCTION)
             (IMPLIES (LISTP R)
                      (LESSP (LNG2 "NIL" (CDR L))
                             (LNG2 L R)))
             NIL NIL)
(PROVE.LEMMA LNG2.EXPR.GOES.DOWN2 (INDUCTION)
             (IMPLIES (LISTP R)
                      (LESSP (LNG2 "NIL" (BEGIN (CDR R)))
                             (LNG2 L R)))
             NIL NIL)
(PROVE.LEMMA LNG2.EXPR.STAYS.EVEN (INDUCTION)
             (IMPLIES (LISTP R)
                      (EQUAL (LNG2 (CONCATCH L
                                                 (CONS (CAR R)
                                                       "NIL"))
                                   (CDR R))
                             (LNG2 L R)))
             NIL NIL)
(PROVE.LEMMA BEGIN.LESSEQP (INDUCTION)
             (IMPLIES (NOT (EQUAL (COUNT (BEGIN X))
                                  (COUNT X)))
                      (LESSP (COUNT (BEGIN X))
                             (COUNT X)))
             NIL NIL)
(DEFN
  EVALEXPR
  (L R)
  (IF
    (NLISTP R)
    (LEFTPAR)
    (IF
      (NLISTP L)
      (IF (NLISTP (REMAIN R))
          (IF (ATOMP (FIRSTCH R))
              (FIRSTCH R)
              (LEFTPAR))
          (EVALEXPR (CONCATCH L (CONS (CAR R)
                                      "NIL"))
                    (CDR R)))
      (IF
        (LEFTPARP (FIRSTCH L))
        (IF
          (OR (AND (CONNECTP (FIRSTCH R))
                   (NLISTP (REMAIN L)))
              (NOT (LEFTPARP (EVALEXPR "NIL" (REMAIN L)))))
          (IF
            (AND (CONNECTP (FIRSTCH R))
                 (NOT (LEFTPARP (EVALEXPR "NIL"
                                                  (BEGIN (REMAIN R)))))
                 (RIGHTPARP (LASTCH R)))
             (G (CONS (FIRSTCH R)
                      (ONEORTWO (EVALEXPR "NIL" (REMAIN L))
                                (EVALEXPR "NIL"
                                          (BEGIN (REMAIN R))))))
             (LEFTPAR))
           (IF (NLISTP (REMAIN R))
               (LEFTPAR)
               (EVALEXPR (CONCATCH L (CHARCHAIN (FIRSTCH R)))
                         (REMAIN R))))
         (LEFTPAR))))
   NIL)
  (PROVE.LEMMA EVALEXPR.OF.UNARY (REWRITE)
            (IMPLIES
              (AND (CONNECTP U)
                   (LISTP OP1)
                   (PLISTP OP1)
                   (NOT (LEFTPARP (EVALEXPR "NIL" OP1))))
              (EQUAL (EVALEXPR (CONS "<" "NIL")
                               (CONS U
                                     (CONCATCH OP1
                                               (CONS ">" "NIL")
                                               )))
                     (G (CONS U (CONS (EVALEXPR "NIL" OP1)
                                      "NIL")))))
            NIL NIL)
  (PROVE.LEMMA EVALEXPR.TO.THE.RIGHT (REWRITE)
            (EQUAL (EVALEXPR "NIL" (CONS "<" R))
                   (EVALEXPR (CONS "<" "NIL")
                             R))
            NIL NIL)
  (DEFN CUT (L R)
        (IF (OR (NLISTP (REMAIN R))
                (IF (LISTP L)
                    (NOT (LEFTPARP (FIRSTCH L)))
                    (ATOMP (FIRSTCH R)))
                (AND (CONNECTP (FIRSTCH R))
                     (NLISTP (REMAIN L)))
                (NOT (LEFTPARP (EVALEXPR "NIL" (REMAIN L)))))
            (CONS L R)
            (CUT (CONCATCH L (CHARCHAIN (FIRSTCH R)))
                 (REMAIN R)))
        NIL)
  (PROVE.LEMMA CUT.HELPS.EVALEXPR (REWRITE)
            (EQUAL (EQUAL (EVALEXPR (CAR (CUT L R))
                                    (CDR (CUT L R)))
                          (EVALEXPR L R))
                   T)
            NIL NIL)
  (PROVE.LEMMA CUT.CONCATCH (REWRITE)
            (IMPLIES (NOT (LEFTPARP (EVALEXPR U L)))
                    (EQUAL (CUT U (CONCATCH L R))
                           (CONS (CAR (CUT U L))
                                 (CONCATCH (CDR (CUT U L))
                                           R))))
           NIL NIL)
(PROVE.LEMMA INCLUP.LESSP (REWRITE)
             (IMPLIES (AND (NOT (EQUAL R1 R2))
                           (INCLUP L1 R1 L2 R2))
                      (LESSP (LNG R2)
                             (LNG R1)))
             NIL NIL)
(PROVE.LEMMA LNG.NOT.EQUAL (REWRITE)
             (IMPLIES (NOT (EQUAL (LNG R1)
                                  (LNG R2)))
                      (NOT (EQUAL R1 R2)))
             NIL NIL)
(PROVE.LEMMA INCLUP.LESSEQP (REWRITE)
             (IMPLIES (AND (EQUAL R2 R2)
                           (NOT (EQUAL (LNG R2)
                                       (LNG R1)))
                           (INCLUP L1 R1 L2 R2))
                      (LESSP (LNG R2)
                             (LNG R1)))
             NIL NIL)
(PROVE.LEMMA BEGIN.CDR.LNG.LESSP (INDUCTION REWRITE)
             (IMPLIES (LISTP X)
                      (LESSP (LNG (BEGIN (CDR X)))
                             (LNG X)))
             NIL NIL)
(PROVE.LEMMA CDRCUT.LESSEQP.STEP (REWRITE)
             (IMPLIES
               (AND (NOT (EQUAL (LNG (CDR (CUT "NIL" R)))
                                (LNG R)))
                    (INCLUP "NIL" R (CAR (CUT "NIL" R))
                                    (CDR (CUT "NIL" R))))
               (LESSP (LNG (CDR (CUT "NIL" R)))
                      (LNG R)))
             NIL NIL)
(PROVE.LEMMA INCLUP.LR.CUTLR (REWRITE)
             (IMPLIES (PLISTP L)
                      (INCLUP L R (CAR (CUT L R))
                                  (CDR (CUT L R))))
             NIL NIL)
(PROVE.LEMMA
  BEGIN.CDR.CDRCUT.BRIDGE
  (REWRITE)
  (IMPLIES (AND (IMPLIES (NOT (EQUAL (LNG (CDR (CUT "NIL" R)))
                                     (LNG R)))
                         (LESSP (LNG (CDR (CUT "NIL" R)))
                                (LNG R)))
                (LESSP (LNG (BEGIN (CDR (CDR (CUT "NIL" R)))))
                       (LNG (CDR (CUT "NIL" R)))))
```

```
          (LESSP (LNG (BEGIN (CDR (CDR (CUT "NIL" R)))))
                 (LNG R)))
  NIL NIL)
(PROVE.LEMMA CDR.CUT.LESSEQP (REWRITE INDUCTION)
             (IMPLIES (NOT (EQUAL (LNG (CDR (CUT "NIL" R)))
                                  (LNG R)))
                      (LESSP (LNG (CDR (CUT "NIL" R)))
                             (LNG R)))
             NIL NIL)
(PROVE.LEMMA LISTP.CDR.CUT (REWRITE)
             (IMPLIES (LISTP R)
                      (LISTP (CDR (CUT L R))))
             NIL NIL)
(PROVE.LEMMA ADD1.LNG.CDR (REWRITE)
             (IMPLIES (LISTP U)
                      (EQUAL (ADD1 (LNG (CDR U)))
                             (LNG U)))
             NIL NIL)
(PROVE.LEMMA EXPRP.IS.TOTAL (INDUCTION)
             (IMPLIES
               (LISTP R)
               (LESSP (LNG (BEGIN (CDR (CDR (CUT "NIL" R)))))
                      (LNG R)))
             NIL NIL)
(DEFN
  EXPRP
  (R DUMMY)
  (IF
    (NLISTP R)
    F
    (IF
      (NLISTP (REMAIN R))
      (ATOMP (FIRSTCH R))
      (IF (LEFTPARP (FIRSTCH R))
          (IF (CONNECTP (FIRSTCH (CDR (CUT "NIL" R))))
              (IF (EXPRP (BEGIN (REMAIN (CDR (CUT "NIL" R))))
                         (CONS ">" (BEGIN DUMMY)))
                  (RIGHTPARP (LASTCH R))
                  F)
              F)
          F)))
  NIL)
(PROVE.LEMMA BEGIN.CONCATCH.LISTP (REWRITE)
             (IMPLIES (LISTP V)
                      (EQUAL (BEGIN (CONCATCH U V))
                             (CONCATCH U (BEGIN V))))
             NIL NIL)
(PROVE.LEMMA CAR.CONCATCH.LISTP (REWRITE)
             (IMPLIES (LISTP A)
                      (EQUAL (CAR (CONCATCH A B))
                             (CAR A)))
             NIL NIL)
```

```
(PROVE.LEMMA CDR.CONCATCH.LISTP (REWRITE)
             (IMPLIES (LISTP A)
                      (EQUAL (CDR (CONCATCH A B))
                             (CONCATCH (CDR A)
                                       B)))
             NIL NIL)
(PROVE.LEMMA CONCATCH.BEGIN.CONS.LASTCH (REWRITE)
             (IMPLIES (EQUAL (LASTCH A)
                             ">")
                      (EQUAL (CONCATCH (BEGIN A)
                                       (CONS ">" B))
                             (CONCATCH A B)))
             NIL NIL)
(PROVE.LEMMA EVALEXPR.TO.THE.RIGHT2 (REWRITE)
             (IMPLIES (LISTP (REMAIN R))
                      (EQUAL (EVALEXPR "NIL" R)
                             (EVALEXPR (CONS (CAR R)
                                             "NIL")
                                       (CDR R))))
             NIL NIL)
(PROVE.LEMMA
  EVALEXPR.EQUAL.LEFTPAR
  (REWRITE)
  (IMPLIES (AND (LISTP (CAR (CUT L R)))
                (LEFTPARP (FIRSTCH (CAR (CUT L R))))
                (LISTP (REMAIN (CAR (CUT L R))))
                (LEFTPARP (EVALEXPR "NIL"
                                    (REMAIN (CAR (CUT L R))))))
           (NOT (LISTP (REMAIN (CDR (CUT L R))))))
  NIL NIL)
(PROVE.LEMMA LISTP.CAR.CUT (REWRITE)
             (IMPLIES (LISTP L)
                      (LISTP (CAR (CUT L R))))
             NIL NIL)
(PROVE.LEMMA III (REWRITE)
             (IMPLIES (CONNECTP X)
                      (EQUAL (EVALEXPR "NIL" (CONS X Z))
                             (EVALEXPR (CONS X "NIL")
                                       Z)))
             NIL NIL)
(PROVE.LEMMA III2 (REWRITE)
             (IMPLIES (CONNECTP X)
                      (EQUAL (EVALEXPR "NIL" (CONS X Z))
                             (LEFTPAR)))
             NIL NIL)
(DEFN BEGINP (X Y)
      (IF (NLISTP X)
          T
          (IF (EQUAL (CAR X)
                     (CAR Y))
              (BEGINP (CDR X)
                      (CDR Y))
```

```
                  F))
         NIL)
   (PROVE.LEMMA BEGINP.CONCATCH (REWRITE)
                (IMPLIES (BEGINP (CONCATCH X Y)
                                 Z)
                         (BEGINP X Z))
                NIL NIL)
   (PROVE.LEMMA INCLUP.IMPLIES.BEGINP (REWRITE)
                (IMPLIES (INCLUP L1 R1 L2 R2)
                         (BEGINP L1 L2))
                NIL NIL)
   (PROVE.LEMMA BEGINP.CONS.CONS.EQUAL (REWRITE)
                (IMPLIES (BEGINP (CONS X XP)
                                 (CONS Y YP))
                         (EQUAL (EQUAL Y X)
                                T))
                NIL NIL)
   (PROVE.LEMMA BEGINP.IS.REFLEXIVE (REWRITE)
                (BEGINP X X)
                NIL NIL)
   (PROVE.LEMMA BEGINP.CONS.EQUAL.CAR (REWRITE)
                (IMPLIES (AND (EQUAL X X)
                              (BEGINP (CONS X Z)
                                      Y))
                         (EQUAL (CAR Y)
                                X))
                NIL NIL)
   (PROVE.LEMMA BEGINP.CAR.EQUAL (REWRITE)
                (IMPLIES (AND (LISTP X)
                              (EQUAL X X)
                              (BEGINP X Y))
                         (EQUAL (CAR Y)
                                (CAR X)))
                NIL NIL)
   (PROVE.LEMMA EQUAL.CAR.CUT.CROK (REWRITE)
                (IMPLIES (AND (LISTP L)
                              (BEGINP L (CAR (CUT L R))))
                         (EQUAL (CAR (CAR (CUT L R)))
                                (CAR L)))
                NIL NIL)
   (PROVE.LEMMA BEGINP.CAR.CUT (REWRITE)
                (BEGINP L (CAR (CUT L R)))
                NIL NIL)
   (PROVE.LEMMA EQUAL.CAR.CAR.CUT (REWRITE)
                (IMPLIES (LISTP L)
                         (EQUAL (CAR (CAR (CUT L R)))
                                (CAR L)))
                NIL NIL)
   (MOVE.LEMMA NEW.CUT.HELPS.EVALEXPR NIL CUT.HELPS.EVALEXPR NIL)
   (PROVE.LEMMA LEFTPARP.EVALEXPR.BRIDGE (REWRITE)
                (IMPLIES (AND (LEFTPARP (EVALEXPR (CAR (CUT L R))
                                                  (CDR (CUT L R))))
                              (EQUAL (EVALEXPR (CAR (CUT L R))
                                               (CDR (CUT L R)))
                                     (EVALEXPR L R)))
                         (EQUAL (EQUAL (EVALEXPR L R)
                                       "<")
                                T))
                NIL NIL)
    (PROVE.LEMMA
      NOT.LEFTPARP.EVALEXPR.BRIDGE
      (REWRITE)
      (IMPLIES (AND (NOT (LEFTPARP (EVALEXPR (CAR (CUT L R))
                                             (CDR (CUT L R)))))
                    (EQUAL (EVALEXPR (CAR (CUT L R))
                                     (CDR (CUT L R)))
                           (EVALEXPR L R)))
               (NOT (EQUAL (EVALEXPR L R)
                           "<")))
      NIL NIL)
    (PROVE.LEMMA
      LEFTPARP.EVALEXPR.CUT
      (REWRITE)
      (IMPLIES (AND (LISTP (CAR (CUT LL RR)))
                    (LEFTPARP (CAR (CAR (CUT LL RR))))
                    (CONNECTP (FIRSTCH (CDR (CUT LL RR))))
                    (LEFTPARP
                      (EVALEXPR "NIL"
                                (BEGIN (REMAIN (CDR (CUT LL RR))))))
                    )
               (EQUAL (EQUAL (EVALEXPR (CAR (CUT LL RR))
                                       (CDR (CUT LL RR)))
                             "<")
                      T))
      NIL NIL)
    (PROVE.LEMMA
      EVALEXPR.CONNECTP.CUT
      (REWRITE)
      (IMPLIES (AND (LISTP L)
                    (NOT (CONNECTP (FIRSTCH (CDR (CUT L R))))))
               (EQUAL (EQUAL (EVALEXPR (CAR (CUT L R))
                                       (CDR (CUT L R)))
                             "<")
                      T))
      NIL NIL)
    (PROVE.LEMMA
      LEFT.ALREADY.CHECKED
      (REWRITE)
      (IMPLIES
        (AND (LEFTPARP (FIRSTCH (CAR (CUT L R))))
             (LISTP (CAR (CDR (CUT L R))))
             (NOT (LEFTPARP (EVALEXPR "NIL"
                                      (BEGIN (CDR (CDR (CUT L R))))))
                  ))
```

```
          (RIGHTPARP (LASTCH R)))
     (NOT (EQUAL (EVALEXPR (CAR (CUT L R))
                           (CDR (CUT L R)))
                 "<")))
   NIL NIL)
 (PROVE.LEMMA EVALEXPR.RIGHTPARP.LASTCH (REWRITE)
              (IMPLIES (AND (LISTP L)
                            (NOT (RIGHTPARP (LASTCH R))))
                       (EQUAL (EQUAL (EVALEXPR L R)
                                     "<")
                              T))
              NIL NIL)
 (MOVE.LEMMA SAMEAS.CUT.HELPS.EVALEXPR (REWRITE)
             CUT.HELPS.EVALEXPR NIL)
 (PROVE.LEMMA EXPRP.EQUAL.EVALEXPR (REWRITE)
              (EQUAL (EXPRP A DUMMY)
                     (NOT (EQUAL (EVALEXPR "NIL" A)
                                 "<")))
              NIL NIL)
 (PROVE.LEMMA EXPRP.IMPLIES.EVALEXPR (REWRITE)
              (IMPLIES (EXPRP A DUMMY)
                       (NOT (EQUAL (EVALEXPR "NIL" A)
                                   "<")))
              NIL NIL)
 (PROVE.LEMMA EXPRP.IMPLIES.EVALEXPR.LEFTPAR (REWRITE)
              (IMPLIES (AND (LISTP X)
                            (EXPRP (CONS "<" X)
                                   DUMMY))
                       (NOT (EQUAL (EVALEXPR (CONS "<" "NIL")
                                             X)
                                   "<")))
              NIL NIL)
 (MOVE.LEMMA NEW.EXPRP.EQUAL.EVALEXPR NIL EXPRP.EQUAL.EVALEXPR
             NIL)
 (PROVE.LEMMA
   EXPRP.CONCATCH.EXPANDS
   (REWRITE)
   (IMPLIES
     (LISTP (CAR (CDR (CUT (CONS "<" "NIL")
                           (CONCATCH X B)))))
     (EQUAL
       (EXPRP (CONS "<" (CONCATCH X B))
              B)
       (AND (EXPRP (BEGIN (CDR (CDR (CUT (CONS "<" "NIL")
                                         (CONCATCH X B)))))
                   (CONS ">" (BEGIN B)))
            (RIGHTPARP (LASTCH (CONS "<" (CONCATCH X B)))))))
   NIL NIL)
 (PROVE.LEMMA
   EXPRP.CONCATCH.EQUAL.COMMON
   (REWRITE)
   (IMPLIES
    (AND (LISTP X)
         (LISTP B)
         (EXPRP (CONS "<" X)
                B))
    (EQUAL (EXPRP (BEGIN (CDR (CDR (CUT (CONS "<" "NIL")
                                        (CONCATCH X B)))))
                  (CONS ">" (BEGIN B)))
           (EXPRP (CONCATCH (CDR (CDR (CUT (CONS "<" "NIL")
                                           X)))
                            (BEGIN B))
                  (CONS ">" (BEGIN B)))))
  NIL NIL)
(PROVE.LEMMA LASTCH.CDR.LISTP.ELIM (REWRITE)
             (IMPLIES (LISTP (CDR U))
                      (EQUAL (LASTCH (CDR U))
                             (LASTCH U)))
             NIL NIL)
(PROVE.LEMMA LASTCH.CUT (REWRITE)
             (IMPLIES (LISTP V)
                      (EQUAL (LASTCH (CUT U V))
                             (LASTCH V)))
             NIL NIL)
(PROVE.LEMMA TWO.ELEMENTS (REWRITE)
             (IMPLIES (AND (LISTP (CAR U))
                           (NLISTP (LASTCH U)))
                      (LISTP (CDR U)))
             NIL NIL)
(PROVE.LEMMA
  REBUILD.EXPRP.FROM.EXPANSION
  (REWRITE)
  (IMPLIES (AND (LISTP X)
                (LISTP (CAR (CDR (CUT (CONS "<" "NIL")
                                      X))))
                (EXPRP (BEGIN (CDR (CDR (CUT (CONS "<" "NIL")
                                             X))))
                       (CONS ">" (BEGIN B)))
                (EQUAL (LASTCH X)
                       ">"))
           (EXPRP (CONS "<" X)
                  B))
  NIL NIL)
(PROVE.LEMMA LISTP.CAR.CDRCUT.CONCATCH (REWRITE)
             (IMPLIES (AND (EXPRP (CONS "<" X)
                                  B)
                           (LISTP X)
                           (LISTP (CAR (CDR (CUT (CONS "<"
                                                       "NIL")
                                                 X)))))
                      (LISTP (CAR (CDR (CUT (CONS "<" "NIL")
                                            (CONCATCH X B))))))
             NIL NIL)
(PROVE.LEMMA INIT.SEG (REWRITE)
                    (IMPLIES (AND (LISTP B)
                                  (EXPRP A B))
                             (NOT (EXPRP (CONCATCH A B)
                                         B)))
                    NIL NIL)
      (PROVE.LEMMA
        INIT.SEG.EVALEXPR.BRIDGE
        (REWRITE)
        (IMPLIES
          (AND (EQUAL (EXPRP A B)
                      (NOT (LEFTPARP (EVALEXPR "NIL" A))))
               (EQUAL (EXPRP (CONCATCH A B)
                             B)
                      (NOT (LEFTPARP (EVALEXPR "NIL" (CONCATCH A B))))
                      )
               (NOT (LEFTPARP (EVALEXPR "NIL" A)))
               (LISTP B))
          (EQUAL (EVALEXPR "NIL" (CONCATCH A B))
                 "<"))
        NIL NIL)
      (MOVE.LEMMA SAMEAS.EXPRP.EQUAL.EVALEXPR (REWRITE)
                 EXPRP.EQUAL.EVALEXPR NIL)
      (PROVE.LEMMA INIT.SEG.EVALEXPR (REWRITE)
                  (IMPLIES (AND (NOT (LEFTPARP (EVALEXPR "NIL" A)))
                                (LISTP B))
                           (EQUAL (EVALEXPR "NIL" (CONCATCH A B))
                                  "<"))
                 NIL
                 (INIT.SEG.EVALEXPR.BRIDGE
                                      SAMEAS.EXPRP.EQUAL.EVALEXPR))
      (DEFN MINUS (A B)
            (IF (NLISTP A)
                A
                (IF (NLISTP B)
                    A
                    (IF (EQUAL (CAR A)
                               (CAR B))
                        (MINUS (CDR A)
                               (CDR B))
                        A)))
            NIL)
      (PROVE.LEMMA MINUS.EQUAL (REWRITE)
                  (EQUAL (EQUAL (MINUS X Y)
                                (MINUS Y X))
                         (EQUAL X Y))
                  NIL NIL)
      (PROVE.LEMMA
        INIT.SEG.MINUS.BRIDGE
        (REWRITE)
        (IMPLIES (AND (NOT (LEFTPARP (EVALEXPR "NIL" A)))
                      (EQUAL B (CONCATCH A (MINUS B A)))
                      (LEFTPARP (EVALEXPR "NIL"
                                                (CONCATCH A (MINUS B A)))))
                (EQUAL (EVALEXPR "NIL" B)
                       (LEFTPAR)))
  NIL
  (USE.NO.LEMMA.TO.GO.FASTER))
(MOVE.LEMMA NEW.INCLUP.IMPLIES.BEGINP NIL INCLUP.IMPLIES.BEGINP
            NIL)
(PROVE.LEMMA III3 (REWRITE)
             (IMPLIES (AND (INCLUP Y R1 X R2)
                           (CONNECTP (CAR Y))
                           (BEGINP Y X))
                      (EQUAL (EVALEXPR "NIL" X)
                             (LEFTPAR)))
             NIL NIL)
(MOVE.LEMMA SAMEAS.INCLUP.IMPLIES.BEGINP (REWRITE)
            INCLUP.IMPLIES.BEGINP NIL)
(DEFN BEGIN2P (X Y)
      (IF (NLISTP X)
          T
          (IF (LISTP Y)
              (IF (EQUAL (CAR X)
                         (CAR Y))
                  (BEGIN2P (CDR X)
                           (CDR Y))
                  F)
              F))
      NIL)
(PROVE.LEMMA BEGIN2P.MINUS.CONCATCH (REWRITE)
             (IMPLIES (BEGIN2P A B)
                      (EQUAL (EQUAL B (CONCATCH A (MINUS B A)))
                             T))
             NIL NIL)
(PROVE.LEMMA INIT.SEG.WITH.BEGIN2P (REWRITE)
             (IMPLIES (AND (NOT (LEFTPARP (EVALEXPR "NIL" A)))
                           (BEGIN2P A B)
                           (LISTP (MINUS B A))
                           (EQUAL B B))
                      (EQUAL (EVALEXPR "NIL" B)
                             "<"))
             NIL
             (INIT.SEG.MINUS.BRIDGE INIT.SEG.EVALEXPR
                                    BEGIN2P.MINUS.CONCATCH))
(PROVE.LEMMA INCLUP.INIT.SEG (REWRITE)
             (IMPLIES (AND (NOT (LEFTPARP (EVALEXPR "NIL" A)))
                           (INCLUP (CONCATCH A (CONS U V))
                                   R1 B R2)
                           (BEGIN2P A B)
                           (LISTP (MINUS B A)))
                      (EQUAL (EVALEXPR "NIL" B)
                             "<"))
             NIL NIL)
(PROVE.LEMMA INCLUP.CONS.INIT.SEG (REWRITE)
                (IMPLIES
                  (AND (NOT (LEFTPARP (EVALEXPR "NIL" A)))
                       (INCLUP (CONS X (CONCATCH A (CONS U V)))
                               R1
                               (CONS Y B)
                               R2)
                       (BEGIN2P A B)
                       (LISTP (MINUS B A)))
                  (EQUAL (EVALEXPR "NIL" B)
                         "<"))
                NIL NIL)
  (PROVE.LEMMA BEGIN2P.LISTP.MINUS.CONCATCH (REWRITE)
               (IMPLIES (AND (BEGIN2P (CONCATCH A B)
                                      C)
                             (LISTP B)
                             (EQUAL A A))
                        (LISTP (MINUS C A)))
               NIL NIL)
  (PROVE.LEMMA INCLUP.CONCATCH.LISTP.MINUS (REWRITE)
               (IMPLIES (AND (INCLUP (CONCATCH A B)
                                     R1 Z R2)
                             (BEGIN2P (CONCATCH A B)
                                      Z)
                             (LISTP B))
                        (LISTP (MINUS Z A)))
               NIL NIL)
  (PROVE.LEMMA INCLUP.CONS.CONCATCH.LISTP.MINUS (REWRITE)
               (IMPLIES (AND (INCLUP (CONS X (CONCATCH A B))
                                     R1
                                     (CONS Y Z)
                                     R2)
                             (BEGIN2P (CONCATCH A B)
                                      Z)
                             (LISTP B))
                        (LISTP (MINUS Z A)))
               NIL NIL)
  (PROVE.LEMMA BEGIN2P.CONCATCH (REWRITE)
               (IMPLIES (AND (BEGIN2P (CONCATCH A B)
                                      C)
                             (EQUAL A A))
                        (BEGIN2P A C))
               NIL NIL)
  (PROVE.LEMMA INCLUP.CONCATCH.BEGIN2P (REWRITE)
               (IMPLIES (AND (INCLUP (CONCATCH P1 Q)
                                     R1 P2 R2)
                             (BEGIN2P (CONCATCH P1 Q)
                                      P2))
                        (BEGIN2P P1 P2))
               NIL NIL)
  (PROVE.LEMMA INCLUP.CONS.CONCATCH.BEGIN2P (REWRITE)
               (IMPLIES (AND (INCLUP (CONS L1 (CONCATCH P1 Q))
                                     R1
```

```
                                   (CONS L2 P2)
                                   R2)
                              (BEGIN2P (CONCATCH P1 Q)
                                       P2))
                         (BEGIN2P P1 P2))
                NIL NIL)
(PROVE.LEMMA BEGIN2P.CAR.EQUAL (REWRITE)
             (IMPLIES (AND (LISTP A)
                           (BEGIN2P A B))
                      (EQUAL (CAR A)
                             (CAR B)))
             NIL NIL)
(PROVE.LEMMA INCLUP.IMPLIES.BEGIN2P (REWRITE)
             (IMPLIES (AND (INCLUP L1 R1 L2 R2)
                           (EQUAL L1 L1))
                      (BEGIN2P L1 L2))
             NIL NIL)
(PROVE.LEMMA INCLUP.CONSL1.CONSL2.BEGIN2P (REWRITE)
             (IMPLIES (AND (INCLUP (CONS L1 P1)
                                   R1
                                   (CONS L2 P2)
                                   R2)
                           (BEGIN2P (CONS L1 P1)
                                    (CONS L2 P2)))
                      (BEGIN2P P1 P2))
             NIL NIL)
(PROVE.LEMMA SUBSUMES.INCLUP.CONSL1.CONSL2.BEGIN2P (REWRITE)
             (IMPLIES (INCLUP (CONS L1 P1)
                              R1
                              (CONS L2 P2)
                              R2)
                      (BEGIN2P P1 P2))
             NIL NIL)
(PROVE.LEMMA SUBSUMES.INCLUP.CONCATCH.BEGIN2P (REWRITE)
             (IMPLIES (INCLUP (CONCATCH A U)
                              R1 B R2)
                      (BEGIN2P A B))
             NIL NIL)
(PROVE.LEMMA SUBSUMES.INCLUP.CONS.CONCATCH.BEGIN2P (REWRITE)
             (IMPLIES (INCLUP (CONS X (CONCATCH A P))
                              R1
                              (CONS Y B)
                              R2)
                      (BEGIN2P A B))
             NIL NIL)
(PROVE.LEMMA INCLUP.EVALEXPR.CUT (REWRITE)
             (IMPLIES
               (AND (LISTP R2)
                    (LEFTPARP (CAR L2))
                    (INCLUP L1 R1 L2 R2)
                    (NOT (LEFTPARP (EVALEXPR "NIL" (CDR L2)))))
               (EQUAL (EQUAL (CUT L1 R1)
                                   (CONS L2 R2))
                              T))
                        NIL
                        (CAR/CDR.ELIM CDR.NLISTP CAR.NLISTP CDR.CONS
                                      CAR.CONS CONS.EQUAL
                                      INCLUP.CDRL1.CDRL2
                                      CAR.CONCATCH.LISTP III3
                                      SAMEAS.INCLUP.IMPLIES.BEGINP
                                      INCLUP.INIT.SEG
                                      INCLUP.IMPLIES.BEGIN2P
                                      INCLUP.CONCATCH.LISTP.MINUS
                                      SUBSUMES.INCLUP.CONCATCH.BEGIN2P
                                      INCLUP.CONS.INIT.SEG
                              SUBSUMES.INCLUP.CONSL1.CONSL2.BEGIN2P
                                      INCLUP.CONS.CONCATCH.LISTP.MINUS
                              SUBSUMES.INCLUP.CONS.CONCATCH.BEGIN2P))
(PROVE.LEMMA CONCATCH.NIL.PLISTP (REWRITE)
             (IMPLIES (PLISTP X)
                      (EQUAL (CONCATCH X "NIL")
                             X))
             NIL NIL)
(MOVE.LEMMA SHUTOFF.INCLUP.CONCATCH NIL INCLUP.CONCATCH NIL)
(PROVE.LEMMA INCLUP.CONS.CONCATCH.BRIDGE (REWRITE)
             (IMPLIES (AND (INCLUP L (CONCATCH D R)
                                     (CONCATCH L D)
                                     R)
                           (EQUAL (CONCATCH L D)
                                  (CONS U D)))
                      (INCLUP L (CONCATCH D R)
                              (CONS U D)
                              R))
             NIL NIL)
(MOVE.LEMMA SAMEAS.INCLUP.CONCATCH (REWRITE)
            INCLUP.CONCATCH NIL)
(MOVE.LEMMA SHUTOFF.INCLUP.CDRL1.CDRL2 NIL INCLUP.CDRL1.CDRL2
            NIL)
(PROVE.LEMMA INCLUP.CONS.CONCATCH (REWRITE)
             (IMPLIES (AND (PLISTP L)
                           (PLISTP D)
                           (EQUAL (CONCATCH L D)
                                  (CONS U D)))
                      (INCLUP L (CONCATCH D R)
                              (CONS U D)
                              R))
             NIL NIL)
(PROVE.LEMMA WHICH.APPLIES.MORE.OFTEN.THAN.CUT.HELPS.EVALEXPR
             (REWRITE)
             (EQUAL (EVALEXPR (CAR (CUT L R))
                              (CDR (CUT L R)))
                    (EVALEXPR L R))
             NIL
             (SAMEAS.CUT.HELPS.EVALEXPR))
```

```
(PROVE.LEMMA
  IMPLIES.EQUAL.ECA.ECB.EQUAL.EA.EB
  (REWRITE)
  (IMPLIES
    (EQUAL
      (EVALEXPR
        (CAR (CUT (CONS "<" "NIL")
                  (CONCATCH D (CONS V
                                    (CONCATCH Z
                                              (CONS ">" "NIL")))
                  )))
        (CDR (CUT (CONS "<" "NIL")
                  (CONCATCH D (CONS V
                                    (CONCATCH Z
                                              (CONS ">" "NIL")))
                  ))))
      (CONS V (CONS (EVALEXPR "NIL" D)
                    (CONS (EVALEXPR "NIL" Z)
                          "NIL"))))
    (EQUAL (EVALEXPR (CONS "<" "NIL")
                     (CONCATCH D
                               (CONS V
                                     (CONCATCH Z
                                               (CONS ">" "NIL"))
                                     )))
           (CONS V (CONS (EVALEXPR "NIL" D)
                         (CONS (EVALEXPR "NIL" Z)
                               "NIL")))))
  NIL
  (WHICH.APPLIES.MORE.OFTEN.THAN.CUT.HELPS.EVALEXPR))
(MOVE.LEMMA SHUT.OFF2.SAMEAS.CUT.HELPS.EVALEXPR NIL
         SAMEAS.CUT.HELPS.EVALEXPR NIL)
(MOVE.LEMMA
  SHUT.OFF.WHICH.APPLIES.MORE.OFTEN.THAN.CUT.HELPS.EVALEXPR NIL
         WHICH.APPLIES.MORE.OFTEN.THAN.CUT.HELPS.EVALEXPR NIL)
(PROVE.LEMMA TREEP.IMPLIES.NOT.LEFTPARP (REWRITE)
             (IMPLIES (AND (TREEP X)
                           (EQUAL X X))
                      (NOT (EQUAL X "<")))
             NIL NIL)
(PROVE.LEMMA TREEP.AND.EQUAL.IMPLIES.NOT.LEFTPARP (REWRITE)
             (IMPLIES (AND (EQUAL ZZ Z)
                           (TREEP Z))
                      (NOT (EQUAL ZZ "<")))
             NIL NIL)
(PROVE.LEMMA
  EQUAL.CUT.CONS.BRIDGE
  (REWRITE)
  (IMPLIES (AND (EQUAL (CUT L1 R1)
                       (CONS (CONS "<" D)
                             (CONS V (CONCATCH Z
                                               (CONS ">" "NIL"))
```

```
                                                )))
                                    (EQUAL (EVALEXPR (CONS "<" D)
                                                     (CONS V
                                                           (CONCATCH Z
                                                                     (CONS ">"
                                                                          "NIL"))))
                                           (CONS V (CONS (EVALEXPR "NIL" D)
                                                         (CONS (EVALEXPR "NIL" Z)
                                                               "NIL")))))
                               (EQUAL (EQUAL (EVALEXPR (CAR (CUT L1 R1))
                                                       (CDR (CUT L1 R1)))
                                             (CONS V
                                                   (CONS (EVALEXPR "NIL" D)
                                                         (CONS (EVALEXPR "NIL" Z)
                                                               "NIL"))))
                                      T))
                  NIL
                  (CAR.CONS CDR.CONS))
                (PROVE.LEMMA EVALEXPR.IS.CORRECT (REWRITE)
                             (IMPLIES (TREEP X)
                                      (EQUAL (EVALEXPR "NIL" (PRINT X))
                                             X))
                             NIL NIL)))
(DECLARE: DONTCOPY
  (FILEMAP (NIL)))
STOP
```

## REFERENCES

1. Robert S. Boyer and J Strother Moore, "A COMPUTATIONAL LOGIC", unpublished paper, SRI International, Menlo Park, California (May 1978).

2. Robert S. Boyer and J Strother Moore, "Mechanizing the Mathematics of Computer Program Analysis using the Logic of Recursive Functions", unpublished paper, SRI International, Menlo Park, California (Dec 1977) (initial version of [1]).

3. R. Boyer and J Moore, "A COMPUTER PROOF OF THE CORRECTNESS OF A SIMPLE OPTIMIZING COMPILER FOR EXPRESSIONS", Technical Report 5, Contract N00014-75-C-0816, SRI Project 4079, Stanford Research Institute, Menlo Park, California (Jan 1977) (NTIS Number AD-A036 121/2WC).

## DISTRIBUTION LIST

The below listing is the official distribution list for the technical, annual, and final reports for Contract N00014-75-C-0816.

| | |
|---|---|
| Defense Documentation Center<br>Cameron Station<br>Alexandria, VA. 22314 | 12 copies |
| Office of Naval Research<br>Information Systems Program<br>Code 437<br>Arlington, VA. 22217 | 2 copies |
| Office of Naval Research<br>Code 102IP<br>Arlington, VA. 22217 | 6 copies |
| Office of Naval Research<br>Branch Office, Boston<br>495 Summer Street<br>Boston, MASS. 02210 | 1 copy |
| Office of Naval Research<br>Branch Office, Chicago<br>536 South Clark Street<br>Chicago, ILL. 60605 | 1 copy |
| Office of Naval Research<br>Branch Office, Pasadena<br>1030 East Green Street<br>Pasadena, CA. 91106 | 1 copy |
| New York Area Office<br>715 Broadway - 5th Floor<br>New York, N.Y. 10003 | 1 copy |
| Assistant Chief for Technology<br>Office of Naval Research<br>Code 200<br>Arlington, VA. 22217 | 1 copy |
| Naval Research Laboratory<br>Technical Information Div.,<br>Code 2627<br>Washington, D.C. 20375 | 6 copies |
| Dr. A. L. Slafkosky<br>Scientific Advisor<br>Commandant of the Marine Corps (Code RD-1)<br>Washington, D.C. 20380 | 1 copy |
| Office of Naval Research<br>Code 455<br>Arlington, VA. 22217 | 1 copy |
| Office of Naval Research<br>Code 458<br>Arlington, VA. 22217 | 1 copy |
| Naval Elec. Laboratory Center<br>Advanced Software Tech. Div.<br>Code 5200<br>San Diego, CA. 92152 | 1 copy |
| Mr. E. H. Gleissner<br>Naval Ship Res. & Dev. Center<br>Computation & Math. Dept.<br>Bethesda, MD. 20084 | 1 copy |
| Captain Grace M. Hopper<br>NAICOM/MIS Planning Branch<br>(OP-916D)<br>Office of Chief of Naval Operations<br>Washington, D.C. 20350 | 1 copy |
| Mr. Kin B. Thompson<br>Technical Director<br>Information Sys. Div. (OP-91T)<br>Office of Chief of Naval Operations<br>Washington, D.C. 20350 | 1 copy |
| Officer-in-Charge<br>Naval Surface Weapons Center<br>Dahlgren Laboratory<br>Dahlgren, VA 22448<br>Attn: Code KP | 1 copy |